M A D

(Micro-Assembler/Disassembler)

MAD consists of two subprograms, the first is the assembler which converts your source code into machine code which can be understood by the 6809 micro-oricessor. The second part of MAD is the dis-assembler which converts machine code stored in RAM back into source code.

The advantages of using MAD over higher level languages such as BASIC, or PASCAL are firstly that the resulting programs run faster than their high level language counterparts, and secondly that they take up a much smaller area in RAM.

The MAD eprom resides at address 0000 and is designed to interface with a debug eprom.

ASSEMBLY

Once the MAD eprom is in place, and the Pegasus turned on, the menu should show "Assembler" as one of the options. Simply type 'A' to invoke the assembler and it will then prompt for a starting address which should be given in the form of a four digit hexadecimal value.

At the beginning of each line, the hexadecimal address at which you are assembling is displayed. To leave the assembler type the (break) button which will return you to the menu. Otherwise the assembler will wait for a source code line.

Entering a Source Line

This is given in the form of a mnemonic, immediately followed by the register-designation if there is one (i.e. a register lable from the list: A,B,C,D,S,U,X, or Y where C represents the condition code register). Then type a space and the assembler will reply "HUH?" if they are an impossible combination, otherwise it will check to see if they are inherent, if so then the next line will be displayed and a source code line will be expected as before.

E.G. FROM BØØØ

BØØØ TSTA (sp)

BØØ1 RTS (sp)

HUH?

BØØ1 RTS (sp)

BØØ2 (break)

(returns to display menu)

If the command was not inherent then the cursor will move to the right and the assembler will wait for an operand. The parts of a source line are as follows:

```
B000          CMPA          EC34
Address   Mnemonic Register-designation   Operand
```

again, end the argument with a space and the assembler will attempt to assemble the instruction and either return "HUH?" or wait for the next source code line.

```
E.G.   B000   LDA    #B0   Load the A accumulator with B0
                            (base 16)
       B002   BTS          Incorrect mnemonic
       HUH?                Assembler returns for correct line
       B002   BITA B300    Bit with contents of location B300
       B005
```

Register-designations

The optional register immediately following the mnemonic can be an accumulator (A,B, or D), and index register (X, or Y), a stack pointer (S, or U), or the condition code register (C).

Note that in a transfer, or exchange instruction there is no register designation, and that the two registers involved comprise the operand.

Operands

Instructions can be one of seven basic types:

Inherent, Immediate, Extended, Direct, Register, Relative, or Indexed.

The type of the instruction specifies the form of operand used.

Inherent

Inherent has no operand, the assembler will not prompt for one. The mnemonic and the optional register designation comprise the complete instruction.

```
E.G.   B000 SYN      .Synchronise to interrupt
       B001 COMA     .Compliment the A accumulator
       B002 SWI3     .Software interrupt 3
       B004
```

Immediate

Immediate addressing is specified by a # at the beginning of the operand. It means that the operand specifies a particular value to be used, not the address at which it can be found.

In all cases an operand must be specified for an immediate instruction. Various commands which are available in most addressing modes are not available in immediate mode as they would be meaningless e.g. STA #30, or COM #16.

The argument need not be a hexadecimal value, it can also be one or two ASCII characters. This is defined by a " immediately following the # and the number of characters should correspond to the size of the register.

i.e. for a 16 bit register use two characters and for an 8 bit register use only one character. Remember that these characters will have the most significant bit set to 0.

```
E.G.  BØØØ LDA #30      . Load the A accumulator with 30 (HEX)
      BØØ2 CWA #20      . And the condition code register with
                        . 20 and wait for interrupt
      BØØ4 LDD #1010    . Load D with 1010 hex
      BØØ7 LDA #"F      . Load A with ASCII character'F'
                          (=46hex)
      BØØ9 LDS #"EG     . Load S with 'EG'
                        . Note that this is a suicidal instruction
      BØØD COM #B000    . Impossible instruction
      HUH?
      BØØD
```

Extended Addressing

Extended addressing defines an absolute address and is signified by a four digit hexadecimal number, which is used as an address. The byte which is stored at that address will be changed or used. See DIRECT, and SET.

```
E.G.  B000 COM  B300      Compliment byte at B300
      B003 LDA  B300      Load A from B300
      B006 STD  1000      Store D at 1000 and 1001 (combined)
      B009
```

Direct Addressing

Direct addressing produces only one postbyte unlike extended addressing which uses two, thus direct addressing is more compact in RAM. The most significant byte of the address is defined by the direct page register. For both direct and extended addressing, type the full four character hexadecimal address then the assembler will compare the most significant

byte of the address with the direct page register defined in the SET command. If the most significant byte corresponds then direct addressing will be used, otherwise extended will be used. If you type a < before the address then direct addressing will be used meaning that you are responsible for the direct page register holding the correct value. Likewise a > sign before the address will force extended addressing regardless of the direct page register.

```
E.C.    B000    LDA     #10
        B002    TFR     A,DP    Put 10 into direct page register
        B004    SET     10      Inform assembler that it can
                                expect the direct page register
                                to be 10
        B004    LDA     1632    Load A from 1632 - extended
        B007    STA     1032    Store A at 1032 - direct
        B009    LDS     >1032   Load A from 1032 - extended
                                (extra byte used unnecessarily)
        B00C    STA     <1632   Store A at 1632 - direct
                                (actually this will access 1032
                                 NOT 1632)
```

Register Addressing

Register addressing is used by the PSH, PUL, TFR, and EXG instructions. Transfer and exchange are both followed by a pair of equally sized registers separated by a comma. It should be noted that the assembler will not detect different sized registers and will attempt to assemble, but the disassembler will show an error if two unequally sized registers are transferred or exchanged.

The operand of Push or Pull instruction consists of a series of varied length registers separated by commas and may not contain the register pointing to the stack being used.

Only the direct page register is referred to by two characters while the rest are represented by a single letter as follows: A,B,C,D,X,Y,S,U,P, or DP

```
E.G.    B000    TFR A,B         Put A into B without changing A
        B002    EXG B,X         Exchange two different sized
                                registers which is impossible.
        B004    EXG D,X         Exchange the content of D, and X
        B006    PULS A,B,U      Pull A,B,and U, off the S stack
        B008    PSHU D,S        Push D, and S onto user stack
        B00A
```

Relative Addressing

Relative addressing is used by the branch instructions, and requires a one or two byte offset for short or long branches respectively. Simply type the address to be branched to as the operand and the assembler will decide upon whether a short or long branch is required depending upon the difference between the current address and the address being branched to.

As for direct/extended decisions the assembler can be forced to use a small branch by a < sign preceding the address, or a long branch can be forced by a > sign.

```
E.G.  B000   BRA   B000    Short branch to B000
      B002   BRA   >B000   Long branch to B000
      B005   BRA    0000   Long branch to 0000
      B008   BRA   <0000   Short branch supposedly to 0000
                           but will actually equal BRA B000
```

Indexed addressing

The assembler uses indexed addressing as specified by Motorola barring a few exceptions. Firstly, as this assembler always assumes a hexadecimal value, the $ used by them is unnecessary but the values A,B, and D(10,11, and 13 in decimal) must be specified as: 0A, 0B, and 0D so that the assembler will recognize them as offsets rather than the registers A,B, and D.

Another difference between MAD indexed modes and Motorola modes is that, PCR for program counter relative is shortened to ,P. The final difference is that you can request 8, or 16 bit postbytes. To force a 16 bit offset to be used, precede the whole operant with a > . To request an 8 bit postbyte a < sign should be used.

```
E.G.  B000   LDA    ,X      Assembler uses 0 bit offset
      B002   LDA    -1,X    Assembler uses 5 bit offset
      B004   LDA   <5F,X     Assembler uses 8 bit offset
      B007   LDA    3333,X  Assembler uses 16 bit offset
      B00B   LDA    ,X      Forces 8 bit offset (i.e, LDA 00,X)
      B00D   LDA   >,X      Forces 16 bit offset(i.e. LDA 0000,X)
```

Assembler Directives

Other than the standard set of mnemonics avaialble in the 6809 there are five assembler directives to give the assembler instructions which are themselves not encoded into RAM but are operated upon instead, these are: FCB, FCC, SET, ORG and DIS.

FCB

FCB (form constant bytes) is used to store hexadecimal values, the final one terminating with a space.

```
E.G.  B005    FCB 27,3B,(sp) Places the two bytes into RAM
      B007                   Note that the commas are automatic
```

FCC

FCC (Form constant characters) is used to put a sequence of characters into RAM, it will echo a "symbol before the string. Note that spaces cannot be created by this command and should be done by FCB 20.

```
E.G.  B000 FCC "Hello      Places characters H,e,l,l, and
      B005                 0 in RAM with most significant
                           bit = 0
```

SET

SET (Set direct page register). This directive is used to inform the assembler where it can assume the direct page register to be. The assembler will use this to decide upon whether it can use direct or extended addressing, as discussed under DIRECT. When the assembler is initially started the direct page register is assumed to be pointing to ØØ, so unless you have defined the direct page register, any addressing of the 00 page should be forced to be extended.

```
E.G.   B000 LDA  0033
```

Remember that the SET command does not affect the direct page register as such, and this can only be done using a register mode instruction.

```
E.G.   B000 SET  B3        Informs the assembler that it can
       B000                assume the direct page register
                           to be B3
```

ORG

ORG (origin) changes the address at which you are assembling without affecting the memory into which you are about to assemble.

```
E.G.   B080 ORG  B000      Stop assembling at B080
       B000                and continue at B000
```

DIS

DIS (Disassemble) enables you to invoke the disassembler without leaving the assembler. The start and end addresses are requested as usual for disassembly, and when the disassembly is finished, or a (break) is typed control is returned to the assembler without changing the address at which you were assembling.

```
E.G.  B000  DIS              Invoke disassembler
      FORM? B020             Start disassembling from B020
      TO?   B020             Stop disassembling after B020
      B020  NEG  00          Output from disassembler
      B000
```

Lables

Stored in MAD, is the complete list of user system monitor subroutines and system RAM allocation as defined elsewhere in the manual. when using these subroutines and RAM locations you are do not need to remember the address at which they reside.

To use these addresses simply type the first four letters of the location or subroutine desired instead of its address.

To use these addresses simply type the first four letters of the location or subroutine desired instead of its address.

```
E.G.  B000  LDA INBU         Load A from input buffer
      B003  JSR ECHO         Echo character
      B006
```

Note as you type this that after each line the label is removed and the hexadecimal address is inserted instead so do not type more than the first four letters. The labels stored in MAD can only be used for immediate direct, extended, and extended-indirect instructions. Contrary to the method shown in the system monitor subroutine list you do not need to jump via a lookup table.

```
i.e. instead of typing JMP  F800
     simply type       JMP  ECHO
```

This means that the labels will rarely used other than in direct mode.

DISASSEMBLY

The Disassembler is used to convert machine code stored in RAM into source code as would have been entered into the assembler with a few minor differences.

To start the disassembler ensure that the MAD EPROM is in place and that "Disassembler" appears in the menu, then type D and it will prompt for start and end addresses to be given in the form of two four digit hexadecimal values.

The first visible differences are that any ASCII characters that have been typed in under immediate mode will appear as hexadecimal. Long and Short branches will be indiscernable other than by examining the differences between the addresses

of two adjacent instructions, anything entered under the FCB, or FCC directives will appear ridiculous instructions, and direct instructions will only give the least significant byte as the address.

The final difference is that in PSH or PUL instructions the registers will appear in a specific order, not that which you specified, and the D register will appear as A,B.

If an impossible instruction is disassembled the mnemonic will appear as *** followed by the hexadecimal equivalent of the first byte of the instruction then the disassembler will attempt disassembling from the next byte onwards.

```
E.G.    FROM?   B000
        TO?     B001
        B000    ***     71
        B001    LDA     ,X+     (cr)
        (returns to menu)
```

Example Program

The following program will repeatedly output alphabets until the (break) key is typed.

```
A
FROM?     B000
B000      LDA     #"A         Load "A" into the A accumulator
B002      BSR     ECHO        Echo the character to the screen
B005      INCA                Increment the character
B006      CMPA    #"Z         Is the end of the alphabet reached
B008      BLS     B002        If not then echo next character
B00A      BRA     B000        Otherwise reset A and continue
B00C      (break)
          (returns to MENU)

Type M to enter Monitor
Type G to start program
Then type B000(.)             (the address of the program)
```